Semantic rules:

1. Type checking on expressions, as well as on assignment and return statements.
2. Any id referred to is defined in the scope where it is used (failure results in the following error messages: undefined local variable, undefined free function, undefined member, undefined class)
3. Function calls are made with the right number and type of parameters. Expressions passed as parameters in a function call must be of the same type as declared in the function declaration.
4. Referring to an array variable should be made using the same number of dimensions as declared in the variable declaration. Expressions used as an index must be of integer type.
5. Circular class dependencies are reported as semantic error.
6. The “.” operator should be used only on variables of a class type. If so, its right operand must be a member of that class. If not, a “undefined member” should be issued.
7. The type of the operand of a return statement must be the same as declared in its function’s return type, as declared in the function’s declaration.

Design:

Used visitor design pattern for AST traversal. There are two phases of traversal. The first phase involves AST traversal while creating symbol table for each scope and collect entries created by children node. At the end of the first phase, all symbol table are constructed. The second phase involves another AST traversal while making several semantic checking such as type checking for expressions.

1. Symbol table creation phase: create a symbol table for each scope during the traversal of the AST.

ProgNode: create a symbol table and add entries of children nodes. Check multiply declared class and functions. Check shadowed inherited members.

ClassDeclNode: create a symbol table and add entries of children nodes. Create an entry for itself. Check multiply declared identifiers.

FuncDeclNode: Create an entry for itself.

FuncDefNode: create a symbol table and add entries of children nodes. Create an entry for itself.

VarDeclNode: Create an entry for itself.

StatBlockNode: create a symbol table and add entries of children nodes. Create an entry for itself. Check multiply declared identifiers. Check usage of undeclared variables.

FParamsNode: Create an entry for itself.

1. MulOpNode: Check type on the both side of operand.

AddOpNode: Check type on the both side of operand.

AssignNode: Check type on the both side of operand.

ReturnNode: check type.

RelExprNode: Check type on the both side of relational operand.

Other Node: set the type and data of the node based on its child’s type and data.

Use of tools:

1. Used visitor design pattern for AST traversal. It helps add new features to existing class without changing it.